L11 - Maps

February 11, 2020

1 Fun Collection Types

1.1 Maps

Java's `HashMap` type lets us set up collections of *key/value pairs* that can make associations between objects.

Note that the `HashMap` has *two* types: one for the key, and one for the value. Searching the `HashMap` is unidirectional; you need the key to retrieve the value - you cannot retrieve the key based on the value.

[1]:
```
HashMap <String, String> contacts = new HashMap<>();

// populate the contact list with some values
contacts.put("Alice", "(111) 111 1111");
contacts.put("Bob", "(222) 222 2222");
contacts.put("Charlie", "(333) 333 3333");

String number = contacts.get("Alice");
System.out.println(number);
```

```
(111) 111 1111
```

1.2 Collections of Primitive Types

Ths generic collection classes can be used with all class types, but what about the *primitive* types: `int`, `boolean`?

Suppose we want an `ArrayList` of `int`. Primitive types are *not* objects! These must be *wrapped* in objects to be stored in a collection!

Wrapper classes exist for all primitive types:

simple type	wrapper class	unwrapper method
int	Integer	`.intValue()`
boolean	Boolean	
char	Character	
float	Float	

In practice, autoboxing and autounboxing mean we often don't have to do this explicitly.

1.3 Writing class documentation

Your own classes should be documented the same way as the library classes are, in order to facilitate maintenance. Other people should be able to use your class without reading its source code.

Include:

- class name
- a comment describing the overal purpose and characteristics of the class
- a version number
- the authors' names
- documentation for each constructor and method

1.3.1 Methods and Constructors

- the name of the method
- return type
- parameter names and types
- author
- a description of the purpose and function of the method
- a description of each parameter
- a description of the value returned

[2]:
```
/**
* This is a dog class
* Lets you pet the dog
* Good bois only
*
* @author turbonoodles
* @version 1.0
*/
```

[ ]:
```
/**
* Pet the Dog object
*
* @param pats number of pets to pet the goodboi
* @return integer of successful pets
*/
public boolean petTheDog( int pats )
{
    pet_dog_here();
}
```

1.4 Visibility modifiers: `public` vs. `private`

Public elements are accessible to objects of other classes. Private elements are accessible only to objects of the same class.

- Fields should not be public.
- Only methods that are intended for use by other classes should be public.
  - Helper methods may not need to be public.

1.4.1 Information Hiding

Data belonging to one object is hiddent from other objects. Information hiding increases *independence* of objects. Independence is important for large systems and maintainability.

Information hiding is one of the major ideas behind object-oriented programming.

1.4.2 Class variables

Class variables are shared between all instances of a class. It belongs to the class and exists independently of any instances. These are designated by the **`static`** keyword. These are accessed by the class name: `Thermometer.boilingPoint`.

This is useful for a field that may need to be the same in all instances; for example, a physical constant like the gravitational acceleration for an object in freefall.

Constants Variables whose values can be *fixed*. Designated by the **`final`** keyword. `final` fields must be set in their declaration or in the constructor.

Setting `static final` is common. Setting this `public` is not a big deal since you can't change it anyway. That *DOES NOT* imply that the concepts are related.

Upper-case names are often used for class constants: `public static final POTATOES = 4;`

1.4.3 Class methods

A static *method* belongs to its *class* rather than an instance. Static methods are invoked via their class name.

Since they are *independent* of instances, the cannot call instance methods and they cannot access fields.

1.5 Midterm nonsense

Question types:

- find and fix an error in a code fragment
  - describe the error
  - to fix, write one or two lines of code
- insert a method that hasn't been implemented
  - names and parameters are provided

- given a code fragment/method, what is returned?
  - "simple methods"
- definitely includes collections
- 7 or 8 questions
  - no trick questions
  - no multiple choice
  - time management
  - try to arrive ahead of time

Look over

- control flow structures
  - for:each
  - while
  - Iterators
  - for loop
- ArrayList
  - methods
- Array
- String class
- being cagey about Map and 2d-arrays
  - might help with a solution but probably not needed

[ ]: